Application Note

SGMII Troubleshooting Guide

Anagha Jagannath, Melissa Chang

ABSTRACT

Serial Gigabit Media Independent Interface (SGMII) is a type of communication interface that connects the Ethernet PHY to the medium access control (MAC). SGMII uses low-voltage differential signaling (LVDS) to receive and transmit data at 10/100/1000/2500 Mbps. There are four data signals in SGMII, two for the TX path and two for RX path. An additional two signals, one for RX CLK and one for TX CLK, can be also used as an option. The advantage that SGMII offers over other MAC interfaces is that SGMII supports gigabit communication and has lower RF emissions.

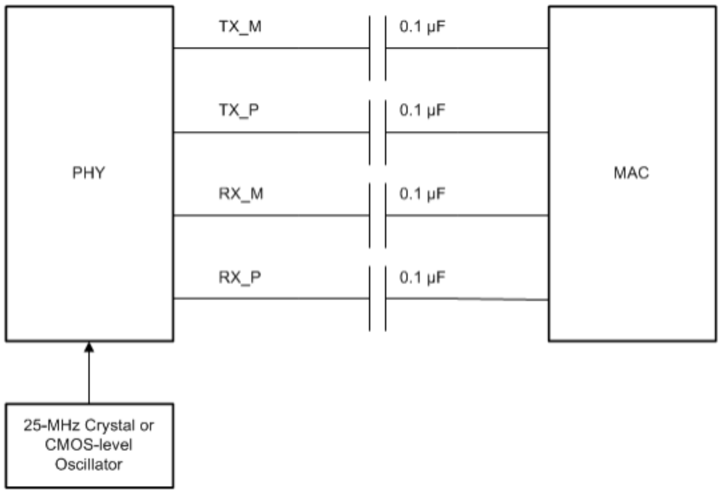


Figure 1-1. Four Wire SGMII Implementation

Table of Contents

Trademarks

All trademarks are the property of their respective owners.

1 Troubleshooting the MAC Interface - SGMII

This guide is intended to troubleshoot common issues, such as link down and packet errors, that can happen while implementing the SGMII MAC interface using TI automotive and industrial Ethernet PHYs.

1.1 Verify Bootstrap Configurations

Before proceeding with any further debug, check that the PHY being used has been correctly strapped into SGMII mode by checking the *Strap Configuration* section of the data sheet.

The strap status of the PHY can be confirmed by reading the CHIP_SOR or SOR_VECTOR register (depending on device).These registers can be accessed using extended register access.

Figure 1-1 is an example of SGMII strap configurations for the DP83TG720S-Q1 Automotive Ethernet PHY. The default MAC Interface of the DP83TG720S-Q1 is SGMII so pins RX_D0, RX_D1, and RX_D2 can be left floating.

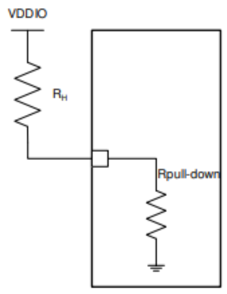


Figure 1-1. 2-level Bootstrap Circuit

1.1.1 SGMII Bootstrap Configuration for DP83TG720S-Q1

Table 1-1. 2-level Bootstraps

Pin Name	Pin No.	Strap Mode	Strap Function
RX_D0	26	1 (default)	MAC[0] = 0
		2	MAC[0] = 1
RX_D1	25	1 (default)	MAC[1] = 0
		2	MAC[1] = 1
RX_D2	24	1 (default)	MAC[2] = 0
		2	MAC[2] = 1

Table 1-2. MAC Interface Selection Bootstraps

MAC[2]	MAC[1]	MAC[0]	Description
0	0	0	SGMII (4-wire)

1.2 Read and Check Register Values

Read the registers and verify that SGMII link is up and SGMII auto negotiation is completed. Note that the initial values of some registers can vary based on strap options.

Shown below are examples of register dumps for the DP83TG720-Q1 and the DP83869. These are possible values for when MDI link is up, the PHY is in SGMII mode, and SGMII link is up.

Table 1-3. DP83TG720S-Q1 Automotive PHY - SGMII Register Values

Register Address	Register Name	Example Value	Description
0x0000	BMCR	0x0140	Bit[14] can be used to configure MII loopback.
0x0001	BMSR	0x0145	Bit[2] indicates MDI link is up.
0x0011	MII_REG_11	0x000B	Bit[11] can be asserted to initiate SGMII soft reset.
0x0428	A2D_REG_40	0x6002	Bits [14:13] indicate output voltage swing is 720mV. Bits can be toggled to increase SGMII output swing.
0x045D	SOR_VECTOR_1	0x0000	Bit[13] indicates SGMII is enabled, Bits [8:6] indicate MAC mode the PHY has been strapped into.
0x0608	SGMII_CTRL_1	0x027B	Bit[0] indicates SGMII auto-negotiation is enabled, Bits [2:1] indicate auto-negotiation timer is 2us. These bits can be toggled to adjust timer.
0x060A	SGMII_STATUS	0x0d46	Bit[11] indicates SGMII link is up, Bit[10] indicates SGMII auto-negotiation is complete.
0x060C	SGMII_CTRL_2	0x001B	Bit[6] can be asserted to restart SGMII auto-negotiation when there is no SGMII link. Bits [5:3] indicate TX and Bits [2:0] indicate RX fifo half full threshold.
0x060D	SGMII_FIFO_STATUS	0x0000	Bit[0] indicates packet underflow. Bit[1] indicates packet overflow.
0x0639	PKT_STAT_1	0x0000	Register value indicates TX packet counter.
0x063A	PKT_STAT_2	0x0000	Register value indicates TX packet counter.
0x063B	PKT_STAT_3	0x0000	TX packet error counter.
0x063C	PKT_STAT_4	0x0000	Register value indicates RX packet counter.
0x063D	PKT_STAT_5	0x0000	Register value indicates RX packet counter.
0x063E	PKT_STAT_6	0x0000	RX packet error counter.

Note

Registers 0x0639, 0x063A, 0x063B are cleared when read in sequence. Similarly, registers 0x063C, 0x063D, 0x063E are also cleared when read in sequence.

Table 1-4. DP83869HM Industrial PHY - SGMII Register Values

Register Address	Register Name	Example Value	Description
0x0000	BMCR	0x1140	Bit[14] can be used to configure MII loopback.
0x0001	BMSR	0x796D	Bit[2] indicates MDI link is up
0x0014	GEN_CFG2	0x29C7	Bit[7] enables SGMII auto-negotiation.
0x0031	GEN_CFG3	0x10B0	Bits [6:5] can be toggled to adjust auto-negotiation timer.
0x0037	SGMII_AUTO_NEG_STATUS	0x0001	Bit[0] indicates SGMII auto-negotiation is complete.

1.3 Auto-Negotiation

Ethernet and SGMII both have an auto-negotiation process that are independent of each other. Ethernet auto-negotiation occurs between an Ethernet PHY and the PHY link partner over the MDI lines. During this process, the two devices exchange information about speed, duplex mode, and flow control and link up at the maximum capability advertised by both link partners.

SGMII auto-negotiation is a process where the PHY sends updated control information to the MAC. This control information is specified in the Cisco SGMII Standard. When the MAC receives this information, the MAC acknowledges reception of the updated control information by asserting an acknowledge bit. In TI Ethernet PHY, this acknowledge bit is associated with the register bit *SGMII Page Received*. In SGMII auto-negotiation, there is no maximization of capabilities, just information exchange between PHY and MAC.

If SGMII link-up issues occur, check that auto negotiation is either disabled in both PHY and MAC or enabled in both PHY and MAC. In either case, the PHY and MAC must support the same communication speeds.

If the PHY comes up before the MAC, a SGMII restart can be required for the MAC to receive control information for successful link up.

If SGMII link up is still unsuccessful, the auto-negotiation timer can also be adjusted for some TI Ethernet PHYs so that the entire auto negotiation cycle is slower or faster on the PHY side. To adjust the timer, set the *sgmii_autoneg_timer* bit field described in the device data sheet and then restart SGMII auto negotiation or reset the PHY by writing 0x1F=0x4000.

1.4 Throughput and Loopback Testing

To verify that the communication errors are SGMII related, a combination of throughput and loop back tests are used, beginning with simple test setups and slowly moving towards more complex and precise test setups. Included are suggestions for how to run these tests for Windows or Linux systems *as an example*, but every system is unique and these methods do not always apply to all designs. For more detailed information on how to run throughput testing with your system, reach out to your Processor or Software vendor.

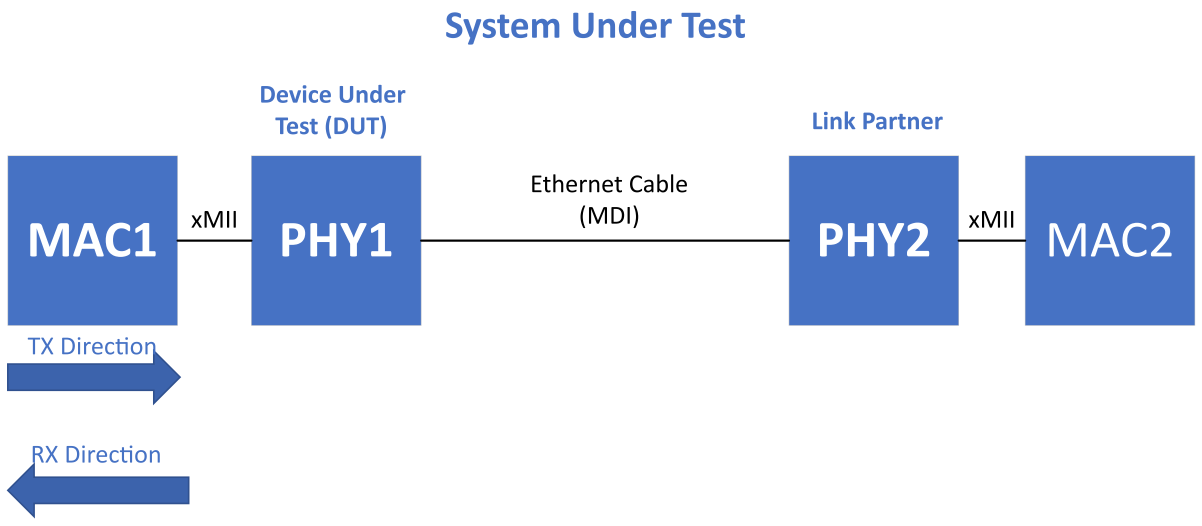


Figure 1-2. Example SGMII Setup

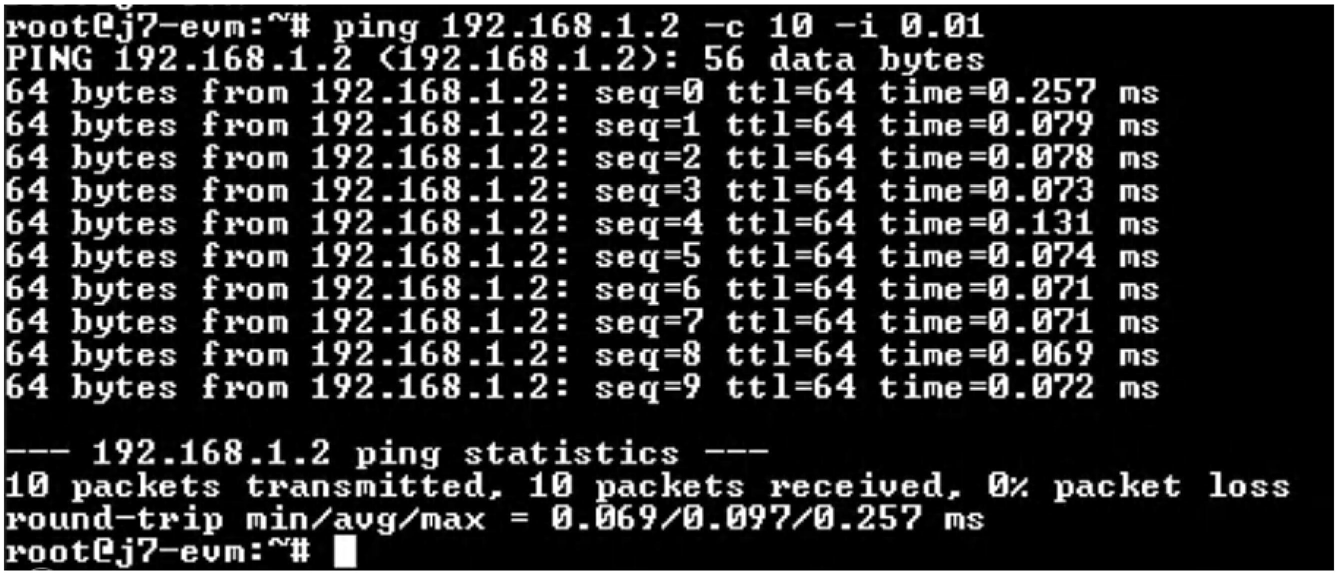
1. **Bidirectional Throughput Testing:** Once SGMII link is up, verify whether there are communication errors by conducting a throughput testing.
2. **RX and TX Throughput Testing:** If there are packet errors, identify if the errors occur in the RX or TX path. The error path can be identified by sending packets in the TX direction only, and checking if the transmitter receives all the correct data. If there are errors, then the issue is with the TX path. Otherwise, the errors come from the RX path.
3. **RX and TX Throughput Testing with Fixed Number of Packets:** If there is packet loss, identify if the loss occurs in the RX or TX path. To identify where packets are being lost, try sending a fixed number of packets in one direction.
4. **Loopback Testing:** Use loopbacks to confirm the issue is within the SGMII block.

1.4.1 Bidirectional Throughput Testing

A simple way of testing throughput is through ping. Ping tests communicate in both the TX and RX direction. The following is an example how to run ping in Windows command prompt or in a Linux terminal:

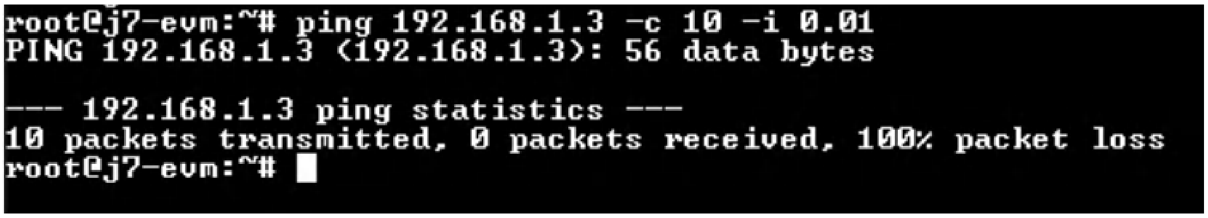
```
ping -c 10 -i 0.5 <IP address> //ping <IP address> 10 times at an interval of 0.5 seconds
```

Figure 1-3 shows an example of a successful ping result and Figure 1-4 shows am example of an unsuccessful ping result in a Linux terminal.



```
root@j7-evm:~# ping 192.168.1.2 -c 10 -i 0.01
PING 192.168.1.2 (192.168.1.2): 56 data bytes
64 bytes from 192.168.1.2: seq=0 ttl=64 time=0.257 ms
64 bytes from 192.168.1.2: seq=1 ttl=64 time=0.079 ms
64 bytes from 192.168.1.2: seq=2 ttl=64 time=0.078 ms
64 bytes from 192.168.1.2: seq=3 ttl=64 time=0.073 ms
64 bytes from 192.168.1.2: seq=4 ttl=64 time=0.131 ms
64 bytes from 192.168.1.2: seq=5 ttl=64 time=0.074 ms
64 bytes from 192.168.1.2: seq=6 ttl=64 time=0.071 ms
64 bytes from 192.168.1.2: seq=7 ttl=64 time=0.071 ms
64 bytes from 192.168.1.2: seq=8 ttl=64 time=0.069 ms
64 bytes from 192.168.1.2: seq=9 ttl=64 time=0.072 ms

--- 192.168.1.2 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 0.069/0.097/0.257 ms
root@j7-evm:~#
```

Figure 1-3. Successful Ping Result



```
root@j7-evm:~# ping 192.168.1.3 -c 10 -i 0.01
PING 192.168.1.3 (192.168.1.3): 56 data bytes

--- 192.168.1.3 ping statistics ---
10 packets transmitted, 0 packets received, 100% packet loss
root@j7-evm:~#
```

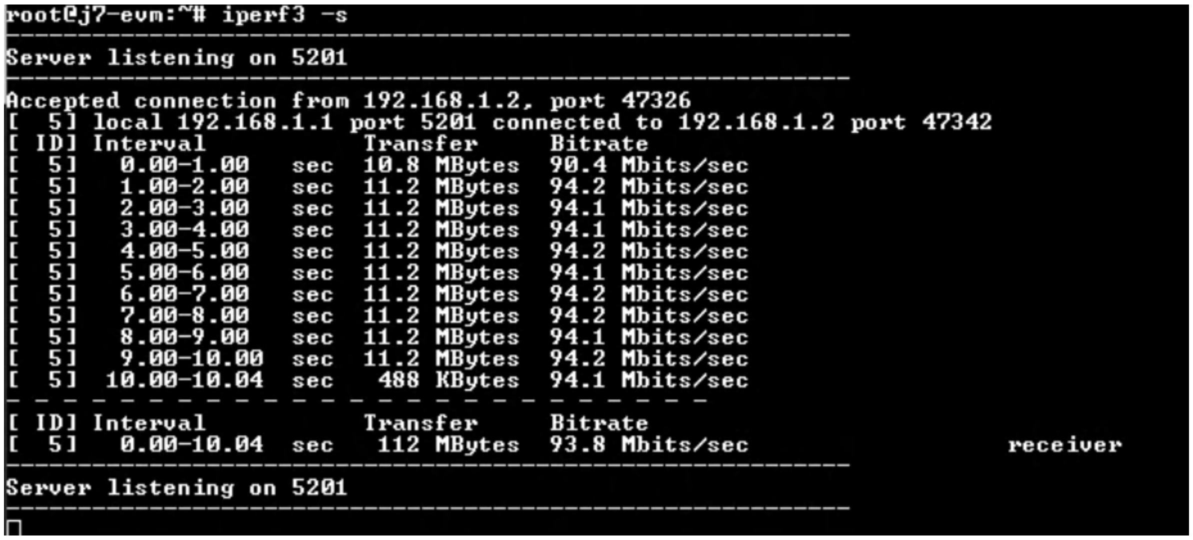
Figure 1-4. Unsuccessful Ping Result

1.4.2 RX and TX Throughput Testing

To run tests with maximized throughput in one or both directions, iPerf3 can be used. IPerf3 can be installed on MAC, Windows, and Linux operating systems. Once installed, iPerf3 can be accessed by entering the directory it has been installed under on either a Terminal or CMD window. A few examples commands with iperf are listed in the following example.

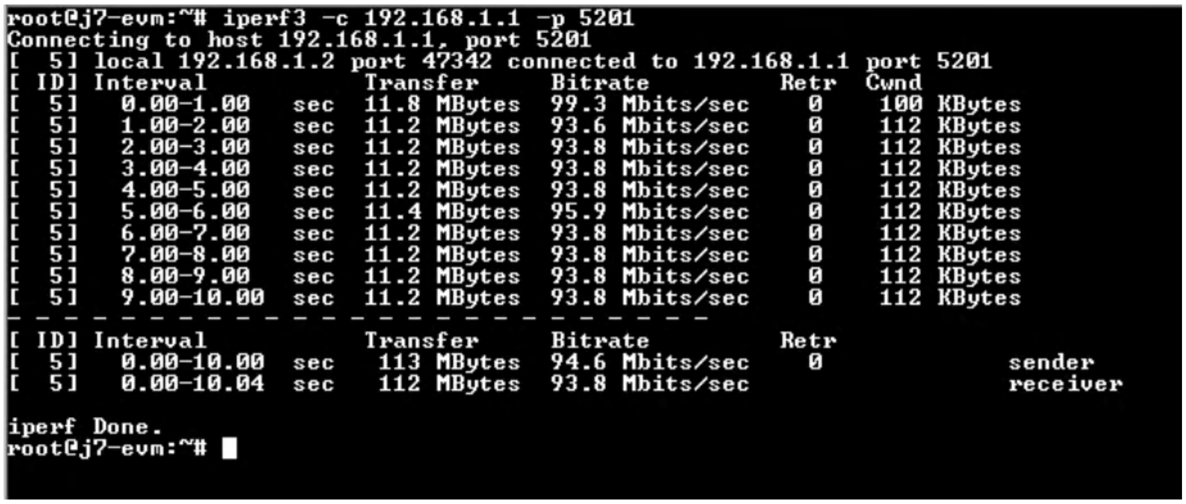
```
C:\Users\labuser\Downloads>cd iperf3.1.1_64

C:\Users\labuser\Downloads>cd iperf3.1.1_64 ipconfig
// ipconfig provides IPv4 address used for ethernet configuration on PC

iperf3 -s // runs iperf as server on one PC

iperf3 -c <server PC IP address> -p <server port number> //runs as client to sniff out packets

iperf3 -h // displays other useful shortcuts and commands
```

Below is an example of a successful transmission of data over 100Mbps with Iperf3 in a Linux terminal. Figure 1-5 shows the server side while Figure 1-6 shows the client side.



```
root@j7-evm:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.1.2, port 47326
[  5] local 192.168.1.1 port 5201 connected to 192.168.1.2 port 47342
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec  10.8 MBytes  90.4 Mbits/sec
[  5]   1.00-2.00   sec  11.2 MBytes  94.2 Mbits/sec
[  5]   2.00-3.00   sec  11.2 MBytes  94.1 Mbits/sec
[  5]   3.00-4.00   sec  11.2 MBytes  94.1 Mbits/sec
[  5]   4.00-5.00   sec  11.2 MBytes  94.2 Mbits/sec
[  5]   5.00-6.00   sec  11.2 MBytes  94.1 Mbits/sec
[  5]   6.00-7.00   sec  11.2 MBytes  94.2 Mbits/sec
[  5]   7.00-8.00   sec  11.2 MBytes  94.2 Mbits/sec
[  5]   8.00-9.00   sec  11.2 MBytes  94.1 Mbits/sec
[  5]   9.00-10.00  sec  11.2 MBytes  94.2 Mbits/sec
[  5]  10.00-10.04  sec   488 KBytes  94.1 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-10.04  sec   112 MBytes  93.8 Mbits/sec                  receiver
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 1-5. Successful Iperf Server Result



```
root@j7-evm:~# iperf3 -c 192.168.1.1 -p 5201
Connecting to host 192.168.1.1, port 5201
[  5] local 192.168.1.2 port 47342 connected to 192.168.1.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  11.8 MBytes  99.3 Mbits/sec    0    100 KBytes
[  5]   1.00-2.00   sec  11.2 MBytes  93.6 Mbits/sec    0    112 KBytes
[  5]   2.00-3.00   sec  11.2 MBytes  93.8 Mbits/sec    0    112 KBytes
[  5]   3.00-4.00   sec  11.2 MBytes  93.8 Mbits/sec    0    112 KBytes
[  5]   4.00-5.00   sec  11.2 MBytes  93.8 Mbits/sec    0    112 KBytes
[  5]   5.00-6.00   sec  11.4 MBytes  95.9 Mbits/sec    0    112 KBytes
[  5]   6.00-7.00   sec  11.2 MBytes  93.8 Mbits/sec    0    112 KBytes
[  5]   7.00-8.00   sec  11.2 MBytes  93.8 Mbits/sec    0    112 KBytes
[  5]   8.00-9.00   sec  11.2 MBytes  93.8 Mbits/sec    0    112 KBytes
[  5]   9.00-10.00  sec  11.2 MBytes  93.8 Mbits/sec    0    112 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec   113 MBytes  94.6 Mbits/sec    0             sender
[  5]   0.00-10.04  sec   112 MBytes  93.8 Mbits/sec                  receiver

iperf Done.
root@j7-evm:~#
```

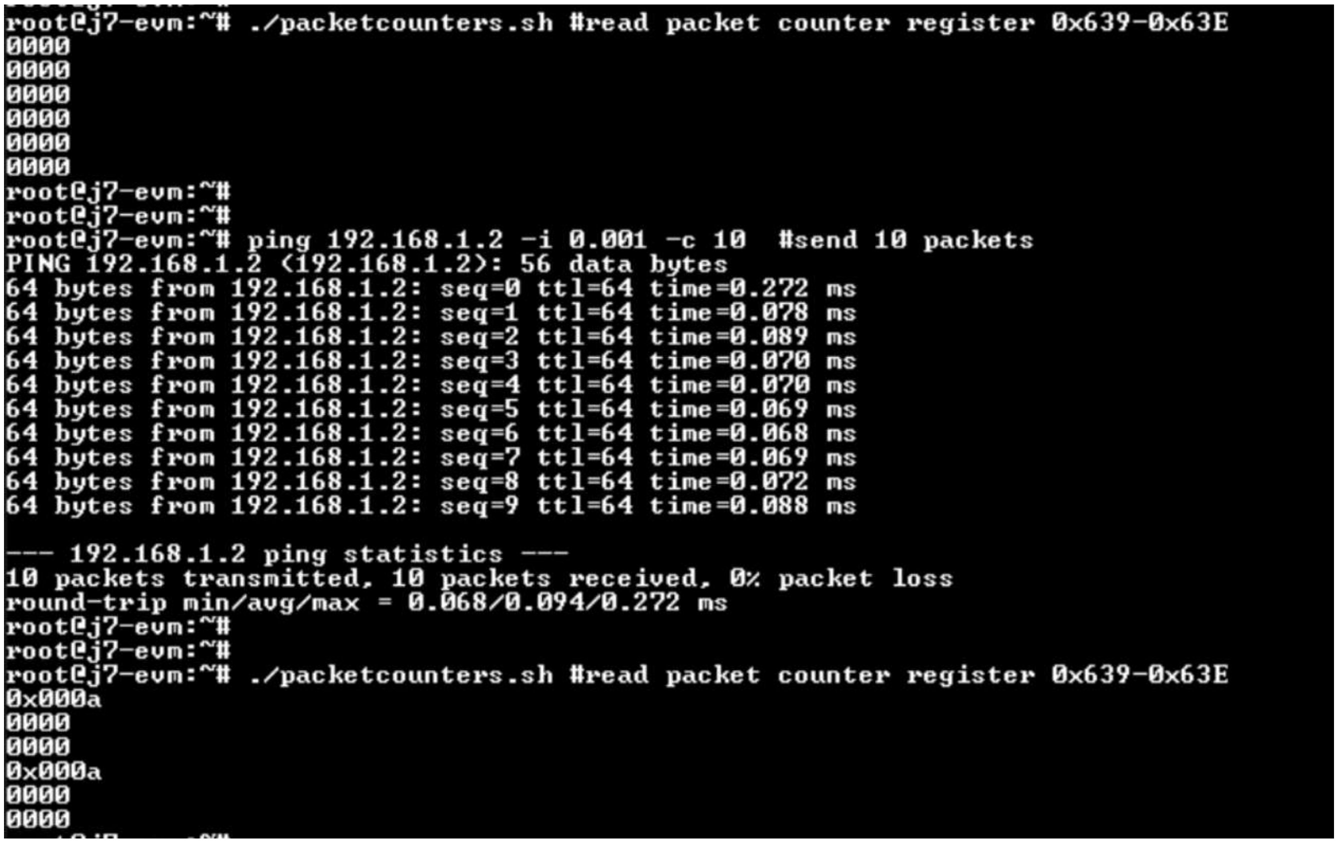
Figure 1-6. Successful Iperf Client Result

1.4.3 RX and TX Throughput Testing with Fixed Number of Packets

For Single Pair Ethernet devices released after 2018, there are RX/TX packet counter and error counter registers that can be found in the data sheet. For example, in the DP83TG720S-Q1 these registers are 0x639-0x63E. Follow these steps to check communication in RX and TX direction:

1. Power on and link up the system under test.
2. Disable any background packets in the system.
3. Read 0x639-0x63E to clear the registers.
4. Send 100 packets across the system in both directions.
5. Check that the TX/RX counters are incrementing correctly and that there are no CRC errors in registers, the registers needs to have the following values: 0x639=0x64, 0x63A=0x0, 0x63B=0x0, 0x63C=0x64, 0x63D=0x0. 0x63E=0x0.

Figure 1-7 shows an example of using DP83TC812's packet counter registers to check the RX and TX communication. The script ./packetcounters.sh prints out registers 0x639-0x63E in order. The register values indicate 10 packets are sent and 10 packets are received in the PHY and there are no errors. Therefore, RX and TX communication are working.

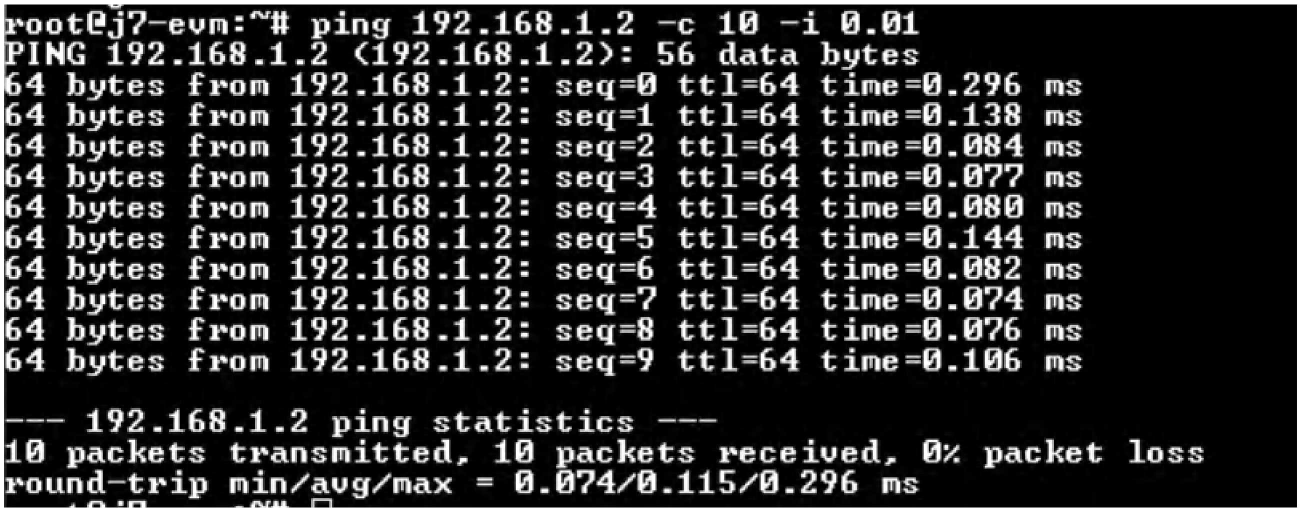


```
root@j7-evm:~# ./packetcounters.sh #read packet counter register 0x639-0x63E
0000
0000
0000
0000
0000
0000
root@j7-evm:~#
root@j7-evm:~#
root@j7-evm:~# ping 192.168.1.2 -i 0.001 -c 10  #send 10 packets
PING 192.168.1.2 (192.168.1.2): 56 data bytes
64 bytes from 192.168.1.2: seq=0 ttl=64 time=0.272 ms
64 bytes from 192.168.1.2: seq=1 ttl=64 time=0.078 ms
64 bytes from 192.168.1.2: seq=2 ttl=64 time=0.089 ms
64 bytes from 192.168.1.2: seq=3 ttl=64 time=0.070 ms
64 bytes from 192.168.1.2: seq=4 ttl=64 time=0.070 ms
64 bytes from 192.168.1.2: seq=5 ttl=64 time=0.069 ms
64 bytes from 192.168.1.2: seq=6 ttl=64 time=0.068 ms
64 bytes from 192.168.1.2: seq=7 ttl=64 time=0.069 ms
64 bytes from 192.168.1.2: seq=8 ttl=64 time=0.072 ms
64 bytes from 192.168.1.2: seq=9 ttl=64 time=0.088 ms

--- 192.168.1.2 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 0.068/0.094/0.272 ms
root@j7-evm:~#
root@j7-evm:~#
root@j7-evm:~# ./packetcounters.sh #read packet counter register 0x639-0x63E
0x000a
0000
0000
0x000a
0000
0000
```

Figure 1-7. Checking Throughput and Packet Counters with DP83TC812

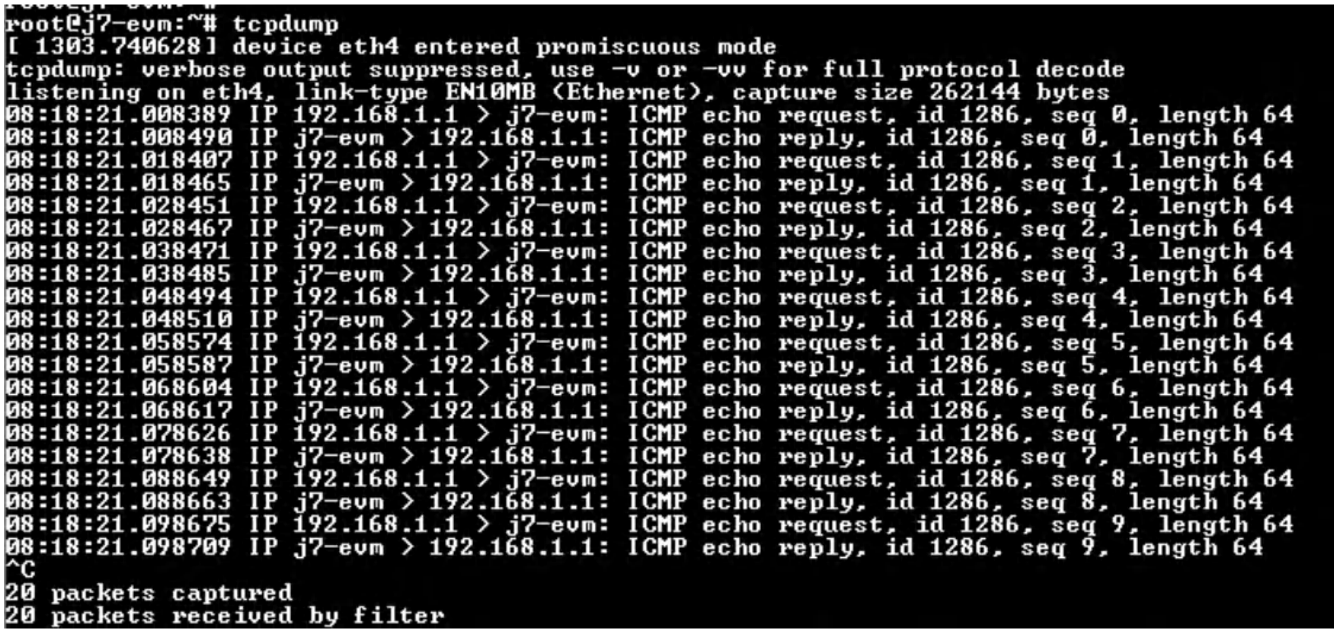
If an Ethernet PHY without packet counter registers is being used or the background packets cannot be disabled in the system, Wireshark or Tcpdump can be used to sniff the packets. Follow these steps to check communication in TX direction:

1. Power on and link up the system under test.
2. Start running tcpdump or wireshark on the link partner's terminal.
3. Ping/send 10 packets from the device under test (DUT) to the link partner. See Figure 1-8.
4. If all 10 packets are received, then there can be 10 requests that show up in wireshark/tcpdump. See Figure 1-9.



```
root@j7-evm:~# ping 192.168.1.2 -c 10 -i 0.01
PING 192.168.1.2 (192.168.1.2): 56 data bytes
64 bytes from 192.168.1.2: seq=0 ttl=64 time=0.296 ms
64 bytes from 192.168.1.2: seq=1 ttl=64 time=0.138 ms
64 bytes from 192.168.1.2: seq=2 ttl=64 time=0.084 ms
64 bytes from 192.168.1.2: seq=3 ttl=64 time=0.077 ms
64 bytes from 192.168.1.2: seq=4 ttl=64 time=0.080 ms
64 bytes from 192.168.1.2: seq=5 ttl=64 time=0.144 ms
64 bytes from 192.168.1.2: seq=6 ttl=64 time=0.082 ms
64 bytes from 192.168.1.2: seq=7 ttl=64 time=0.074 ms
64 bytes from 192.168.1.2: seq=8 ttl=64 time=0.076 ms
64 bytes from 192.168.1.2: seq=9 ttl=64 time=0.106 ms

--- 192.168.1.2 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 0.074/0.115/0.296 ms
```

Figure 1-8. Sending 10 Packets from DUT to Link Partner



```
root@j7-evm:~# tcpdump
[ 1303.740628] device eth4 entered promiscuous mode
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth4, link-type EN10MB (Ethernet), capture size 262144 bytes
08:18:21.008389 IP 192.168.1.1 > j7-evm: ICMP echo request, id 1286, seq 0, length 64
08:18:21.008490 IP j7-evm > 192.168.1.1: ICMP echo reply, id 1286, seq 0, length 64
08:18:21.018407 IP 192.168.1.1 > j7-evm: ICMP echo request, id 1286, seq 1, length 64
08:18:21.018465 IP j7-evm > 192.168.1.1: ICMP echo reply, id 1286, seq 1, length 64
08:18:21.028451 IP 192.168.1.1 > j7-evm: ICMP echo request, id 1286, seq 2, length 64
08:18:21.028467 IP j7-evm > 192.168.1.1: ICMP echo reply, id 1286, seq 2, length 64
08:18:21.038471 IP 192.168.1.1 > j7-evm: ICMP echo request, id 1286, seq 3, length 64
08:18:21.038485 IP j7-evm > 192.168.1.1: ICMP echo reply, id 1286, seq 3, length 64
08:18:21.048494 IP 192.168.1.1 > j7-evm: ICMP echo request, id 1286, seq 4, length 64
08:18:21.048510 IP j7-evm > 192.168.1.1: ICMP echo reply, id 1286, seq 4, length 64
08:18:21.058574 IP 192.168.1.1 > j7-evm: ICMP echo request, id 1286, seq 5, length 64
08:18:21.058587 IP j7-evm > 192.168.1.1: ICMP echo reply, id 1286, seq 5, length 64
08:18:21.068604 IP 192.168.1.1 > j7-evm: ICMP echo request, id 1286, seq 6, length 64
08:18:21.068617 IP j7-evm > 192.168.1.1: ICMP echo reply, id 1286, seq 6, length 64
08:18:21.078626 IP 192.168.1.1 > j7-evm: ICMP echo request, id 1286, seq 7, length 64
08:18:21.078638 IP j7-evm > 192.168.1.1: ICMP echo reply, id 1286, seq 7, length 64
08:18:21.088649 IP 192.168.1.1 > j7-evm: ICMP echo request, id 1286, seq 8, length 64
08:18:21.088663 IP j7-evm > 192.168.1.1: ICMP echo reply, id 1286, seq 8, length 64
08:18:21.098675 IP 192.168.1.1 > j7-evm: ICMP echo request, id 1286, seq 9, length 64
08:18:21.098709 IP j7-evm > 192.168.1.1: ICMP echo reply, id 1286, seq 9, length 64
^C
20 packets captured
20 packets received by filter
```

Figure 1-9. Example TCP Dump Log

The steps for testing the RX direction are similar except the role of the link partner and DUT are reversed.

1.4.4 Loopback Testing

Loopback tests can confirm any errors are a result of the SGMII connection. If errors occur during the MII loopback, then the issue is a result of the MII interface. If errors occur during reverse loopback then the issue is a result of the MDI interface.

1.4.4.1 MII Loopback

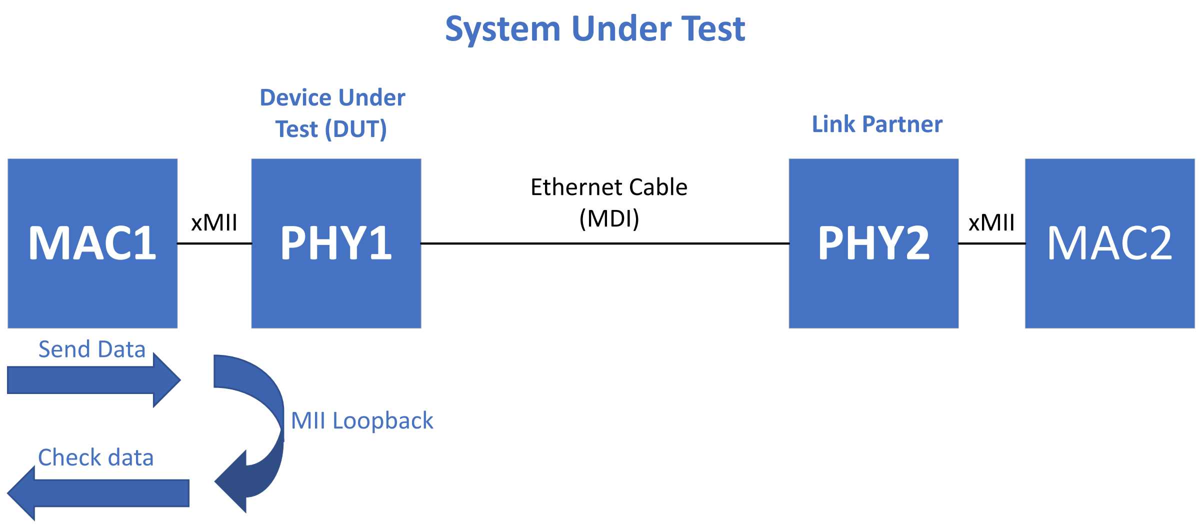
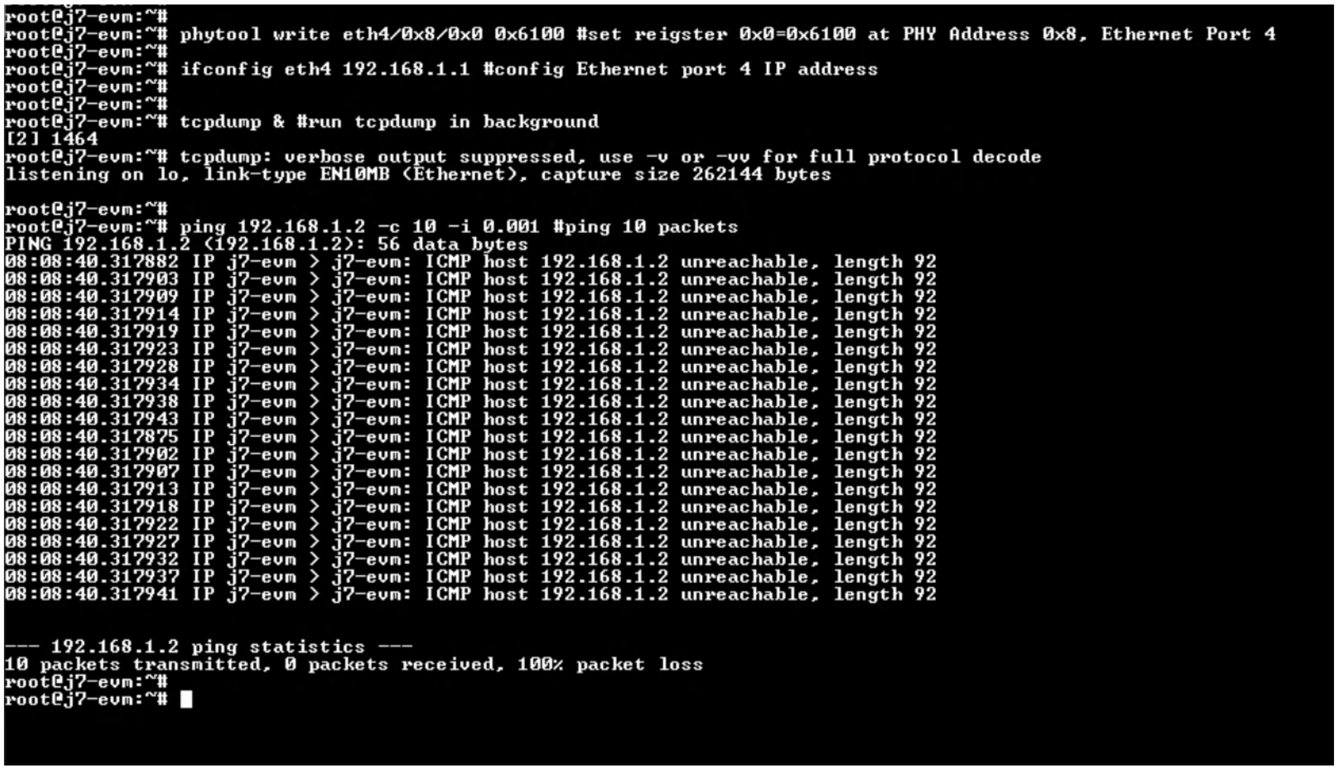


Figure 1-10. MII Loopback Diagram

1. Power on the device under test
2. Enable MII loopback in register 0x0
3. Send a fixed number of packets from the MAC. If ping is used to send packets, do not use the same IP address as the source because the packets are never transferred to the PHY. Use a random IP address.
4. If all the packets sent are also received, then the SGMII connection is working. Tcpdump or Wireshark can be used to check the number of packets received.

In Figure 1-11, 10 packets are output from the MAC after MII loopback in enabled. Ping shows that 10 packets are sent and 0 packets are received. This is expected because the destination address is nonexistent. However, the TCPdump log shows all of the outbound in inbound packets. There are 20 messages total. 10 messages are outgoing and then sent back the the host through loopback. Therefore, the MII block within the PHY is working.



```
root@j7-evm:~#
root@j7-evm:~# phytool write eth4/0x8/0x0 0x6100 #set reigster 0x0=0x6100 at PHY Address 0x8, Ethernet Port 4
root@j7-evm:~#
root@j7-evm:~# ifconfig eth4 192.168.1.1 #config Ethernet port 4 IP address
root@j7-evm:~#
root@j7-evm:~#
root@j7-evm:~# tcpdump & #run tcpdump in background
[2] 1464
root@j7-evm:~# tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes

root@j7-evm:~#
root@j7-evm:~# ping 192.168.1.2 -c 10 -i 0.001 #ping 10 packets
PING 192.168.1.2 (192.168.1.2): 56 data bytes
08:08:40.317882 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317903 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317909 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317914 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317919 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317923 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317928 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317934 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317938 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317943 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317875 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317902 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317907 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317913 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317918 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317922 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317927 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317932 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317937 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92
08:08:40.317941 IP j7-evm > j7-evm: ICMP host 192.168.1.2 unreachable, length 92


--- 192.168.1.2 ping statistics ---
10 packets transmitted, 0 packets received, 100% packet loss
root@j7-evm:~#
root@j7-evm:~# █
```

Figure 1-11. Testing MII Loopback in Linux

To shows only the outgoing or incoming packets, use the following commands:

```
tcpdump dst hst <host IP> & #Packets coming to the host
tcpdump src hst <host IP> & #Packets sent out from the host
```

Note

MII Loopback does note increment the packet counters in register 0x639-0x63E in TI SPE Ethernet PHYs. Use digital loopback instead as a loopback option to increment these counters.

1.4.4.2 Reverse Loopback

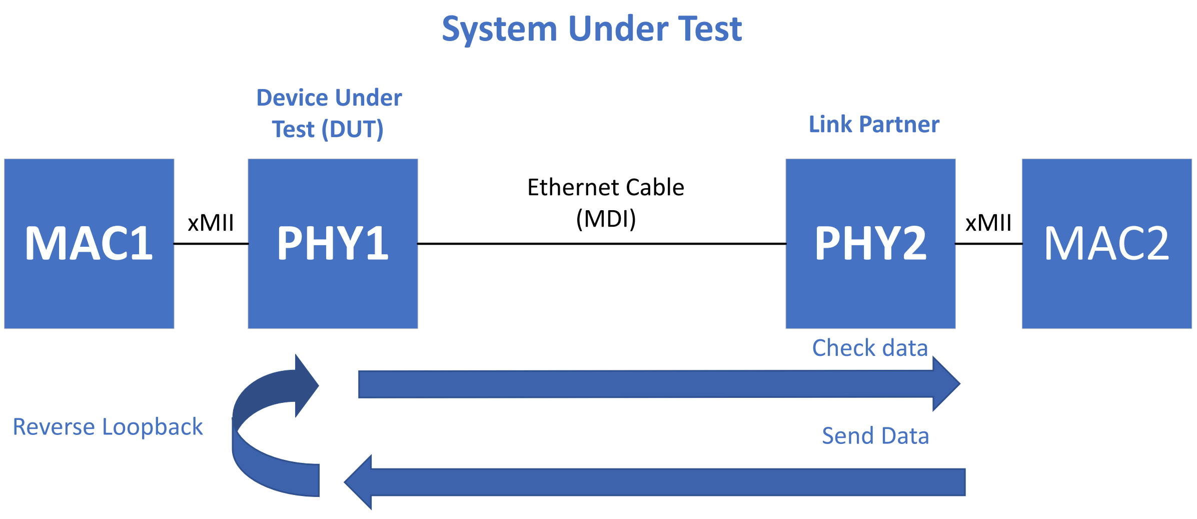


Figure 1-12. Reverse Loopback Diagram

1. Power on the system under test.
2. Enable reverse loopback in register 0x16.
3. Send a fixed number of packets from the PHY's MDI link partner. If ping is used, do not use the same IP address as the source.
4. If all the packets sent are also received, then the MDI connection is working.

1.5 Check the Clock Signal

Using a clock source that fails to meet the PPM specifications mentioned in the data sheet results in issues with register access, packet loss, link down, or complete in-operation of a PHY.

Reach out to the clock source vendor for a crystal report and measure the PPM directly to confirm the clock source being used meets the correct PPM specifications.

To measure the PPM of the clock source, measure the average frequency of the CLK_OUT pin of the PHY with a signal analyzer.

Note

Do not probe XI/XO pins directly to measure the PPM because these pins are more sensitive and can be affected by the capacitance of the probes.

If the average frequency measured by the signal analyzer is f_{avg}, the PPM can be calculated with the following equation:

$$[(f_{expected} - f_{avg}) / f_{expected}] \times 10^6 \quad (1)$$

For instance, if the requirement is 25MHz±100ppm, the average frequency must be in the range of 24.997500MHz - 25.002500MHz.

1.6 Measure the SGMII Eye

The following section details the eye mask requirements for SGMII.

1.6.1 SGMII Eye Mask Requirements

To measure the SGMII output of the PHY, use an oscilloscope to measure near the input pins.

The *input* to the TX_M and TX_P pins on the PHY needs to follow the minimum requirements shown in Figure 1-13.

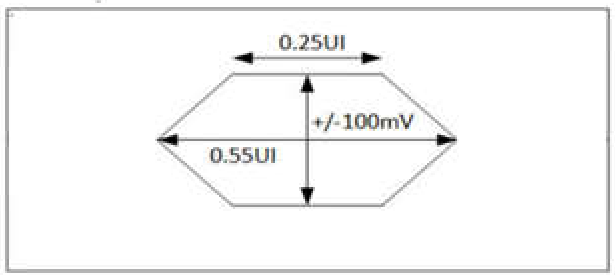


Figure 1-13. SGMII Eye Mask Input Requirement

Section 1.6.1 is a sample eye measured against the input requirement, with the y-axis measured in millivolts and the x-axis measured in ns. Since the mask fits in this eye, the mask meets SGMII input requirements.

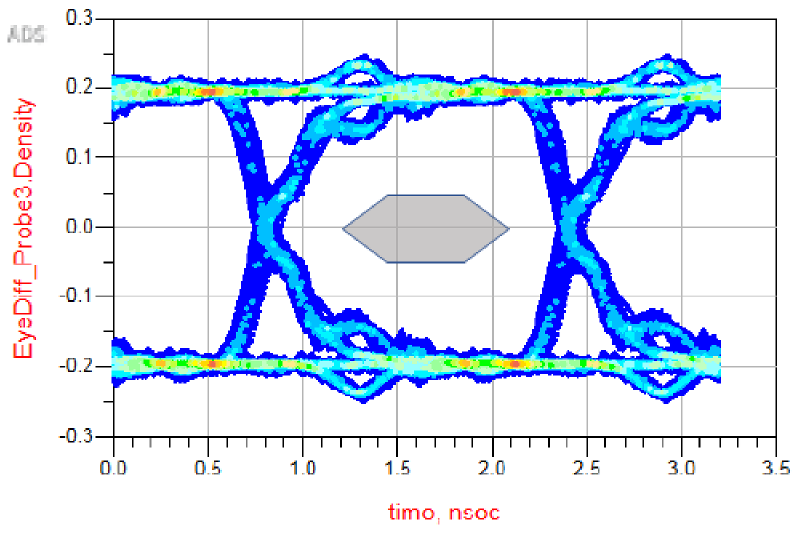


Figure 1-14. SGMII Example Simulation

If the PHY SGMII input does not meet the requirements shown above, there can be some issues with the layout. See Section 1.7 for more details. The PHY SGMII output can also be adjusted with the VOD swing registers.

For example, these settings can be changed by toggling bits 13 and 14 of register 0x428 in the DP83TG720S-Q1.

Table 1-5. DP83TG720S-Q1: 428h Output Swing Settings

Bits 14-13	Register Value	SGMII Output Swing	Unit
00b	0x0002	1000	mV
01b	0x2002	1260	mV
10b	0x4002	900	mV
11b	0x6002	720	mV

1.7 SGMII Layout

Another cause of packet loss, errors, and link down can be length mismatches and discontinuities in SGMII traces. Traces needs to have minimal bends and RX traces needs to be length matched to RX traces and TX traces needs to be length matched to TX traces to make sure of proper transmission of data and successful link up every time the system is powered on.

Additionally, make sure that 100nf AC coupling capacitors have been placed close to the transmitter pins. There needs to also be differential impedance of 100Ω present across the traces and the traces need to be less than 6 inches.

A full layout checkout can be found in the schematic checklist found Design Tools and Simulation section of each product page.

2 Summary

This application note provides a suggested flow for troubleshooting common SGMII issues. If issues with SGMII persist after following the pointers listed previously, please reach out to a TI representative for further guidance on TI E2E Designs Support Forums.

3 References

- Texas Instruments, DP83TG720 Product Folder.
- Texas Instruments, DP83869 Product Folder.
- Cisco SGMII Standard.